REMARKS

This Amendment is filed in response to the Office action dated July 16, 2009, with Request for Continued Examination (RCE) and a 2-month extension of time filed herewith on even date. All objections and rejections are respectfully traversed.

Claims 1 – 15, 19 – 20, and 23 – 42 are pending in this case.

Claim Rejection – 35 USC §103

At paragraphs 3 – 26 of the Office Action claims 1, 3, 6 – 10, 13 – 15, 19, 20, 23, 25, 28 – 31, 33, and 36 – 42 were rejected under 35 U.S.C §103(a) as being unpatentable over Federwisch et al., U.S. Patent Publication No. 2003/0182313 (hereinafter "Federwisch"), in view of Edwards, U.S. Publication No. 2003/0182389 (hereinafter "Edwards").

The present invention, as set forth in representative claim 1, comprises:

1. A method for operating a data storage system, comprising:

creating a writable virtual disk (vdisk) at a selected time, the writable vdisk referencing changes in data stored in the data storage system after the writable vdisk was created;

maintaining a backing store, the backing store referencing data stored in the data storage system which has not been changed since the writable vdisk was created;

loading blocks of the writable vdisk from a disk into a memory, the loaded blocks including a writable vdisk indirect block having a plurality of fields, each field storing a valid pointer to a data block or an invalid pointer representing a particular hole of a plurality of holes, ***where each hole instructs the data storage system to examine a corresponding virtual block number pointer in the backing store***;

loading blocks of the backing store from a disk into the memory, the loaded blocks including a backing store indirect block having a plurality of fields, each backing store indirect block field corresponding to a field of the writable vdisk indirect block, one or more backing store indirect block fields having a pointer to a data block;

searching each field of the writable vdisk indirect block for a hole; and

replacing each field having a hole in the writable vdisk indirect block with a new pointer to the data block referenced by the corresponding backing store indirect block field to update the writable vdisk to reference both the data which is unchanged since the writable vdisk was created and the data which has been changed since the writable vdisk was created.

Federwisch discloses a system and method "for remote asynchronous replication or mirroring of changes in a source file system snapshot in a destination replica file system using a scan (via a scanner) of the blocks that make up two versions of a snapshot of the source file system, which identifies changed blocks in the respective snapshot files…" *See* Federwisch, paragraph [0014]. Further, "[w]here a destination replicated snapshot may be needed at the source to, for example, rebuild the source qtree snapshot, (in other words, the role of the source and destination snapshot are reversed) the use of generalized rollback requires that the inode map be properly related between source and destination." *See* Federwisch, paragraph [0128]. "However, the inode map residing on the destination does not efficiently index the information in a form convenient for use by the source. Rather, the source would need to hunt randomly through the order presented in the map to obtain appropriate values." *See* Federwisch, paragraph [0128]. Thus, Federwisch performs "a 'flip' of map entries." *See* Federwisch, paragraph [0129]. Specifically, "the destination and source negotiate to transfer the inode map file to the source from the destination." *See* Federwisch, paragraph [0129]. "Next the source (which after the negotiation becomes the new destination), creates an empty inode map file with one entry for each inode of the source qtree." *See* Federwisch, paragraph [0130]. "[T]he new destination looks up the Nth inode from the entries associated with the old destination in the stored inode map file (i.e. the map from the old destination/new source)." (Emphasis added) *See* Federwisch, paragraph [0131]. If an entry in the old destination do not exist, then a "zero entry is created in the new inode map file, **representing that the Nth inode of the new source (old destination) is not allocated**." (Emphasis added). *See* Federwisch, paragraph [0131]. However, if there is an entry in the old destination, "a new entry [is created] in the new inode map file. The new entry maps the new source (old destination) Nth inode to the proper new destination (old source) inode." (Emphasis added). *See* Federwisch, paragraph [0131].

Edwards discloses a system and method for "performing an on-line checking [of] a file system in which inodes and directories comprising the file system are checked when first accessed." *See* Edwards, paragraph [0013]. Specifically, "[t]o check an inode, the buffer trees associated with the inode are verified in accordance with procedure 800

shown in FIG. 8. This procedure works by traversing the various branches of the buffer tree and verifying certain key items. First, in step 805, the inode check verifies that all pointers in the buffer tree are valid. If a pointer is directed to an invalid block, the pointer is cleared." *See* Edwards, paragraph [0048]. An invalid pointer is "a pointer [that] points to an invalid inode or file data block." *See* Edwards, paragraph [0054]. Specifically, the checking procedure in Edwards "ensures that all buffer trees have valid pointers, that any given block does not have multiple points to it, and other file system coherency checks." *See* Edwards, paragraph [0014]; *See also* Edwards, paragraph [0049].

The Applicant respectfully urges that a combination of Federwisch and Edwards does not teach or suggest the Applicant's claimed and "***searching each field of the writable vdisk indirect block for a hole***."

First, there appears to be agreement that Federwisch fails to address this aspect of the Applicant's claim. Specifically, the Office Action states, "Federwisch does not disclose searching each field of the writable vdisk indirect block for a hole." *See* Office Action, page 4.

Second, the Applicant respectfully submits that the deficiencies of Federwisch are not remedied by a combination with Edwards. The Applicant notes that the Office Action suggests that paragraph [0048] and element 805 of Figure 8 of Edwards addresses this feature of the Applicant's claim. The Applicant respectfully requests reconsideration.

The Applicant notes that the in the Applicant's claimed technique, each field of the writable vdisk indirect block is searched for a "**hole**". The Applicant notes that as defined in the Applicant's Specification, "a valid VBN pointer has a non-zero value that directly references a data block, whereas an invalid VBN pointer has **a zero value that represents a hole**. Such a hole instructs the file system to examine the value of the VBN pointer in the corresponding level 1 buffer of the backing store." (Emphasis added). *See* Applicant's Specification, page 25, lines 17 – 20. Thus, the Applicant's claimed technique searches a writable vdisk for holes (e.g., zero values). These zero values or

holes instruct the file system to examine the value of the VBN pointer in a corresponding indirect block of the backing store.

In contrast, Edwards describes a technique that traverses "the various branches of the buffer tree and ... verifies that all pointers in the buffer tree are valid." *See* Edwards, paragraph [0048]. An invalid pointer as defined in Edwards is "a pointer [that] points to an invalid inode or file data block." (Emphasis added) *See* Edwards, paragraph [0054]. Specifically, the checking procedure in Edwards "ensures that all buffer trees have valid pointers, that any given block does not have multiple points to it, and other file system coherency checks." *See* Edwards, paragraph [0014]; *See also* Edwards, paragraph [0049]. The Applicant notes that Edwards's verification process is **not** akin to the Applicant's technique that searches a writable vdisk for holes (e.g., zero values) because an invalid pointer in Edwards does not hold a zero value. Instead, Edwards' invalid pointer is a pointer that points to an invalid inode or file data block. Said differently, Edwards does not search for zero values within its buffer tree. As such, the Applicant respectfully submits that Edwards may not fairly be interpreted to teach or suggest the Applicant's claimed "***searching each field of the writable vdisk indirect block for a hole***."

The Applicant also respectfully urges that a combination of Federwisch and Edwards does not teach or suggest the Applicant's claimed " ... ***where each hole instructs the data storage system to examine a corresponding virtual block number pointer in the backing store***" and "***replacing each field having a hole in the writable vdisk indirect block with a new pointer to the data block referenced by the corresponding backing store indirect block field to update the writable vdisk to reference both the data which is unchanged since the writable vdisk was created and the data which has been changed since the writable vdisk was created***."

In the Applicant's claimed technique, the occurrence of a hole (e.g., zero value) instructs the system to examine a corresponding virtual block number pointer in the backing store and thus **replace the hole in the writable vdisk with a new pointer to the data block** that was referenced by the corresponding backing store indirect block field.

The Applicant respectfully submits that Federwisch fails to address these limitations of the Applicant's claim. The Applicant notes that the Office Action suggests that paragraphs [0131] – [0132] and element 1740 of Figure 17 of Federwisch address this aspect of the Applicant's claim. *See* Office Action, page 4. The Applicant respectfully requests reconsideration.

Specifically, the Applicant notes that paragraphs [0131] – [0132] and Figure 17 of Federwisch describe an Inode Map Flip. *See* Federwisch, Paragraph [0127].

More Specifically, Federwisch states:

> [0128] Where a destination replicated snapshot may be needed at the source to, for example, rebuild the source qtree snapshot, (in other words, the role of the source and destination snapshot are reversed) the use of generalized rollback requires that the inode map be properly related between source and destination. This is because the source inodes do not match the destination inodes in their respective trees.
>
> [0129] One way to provide a source-centric inode map is to perform a "flip" of map entries. FIG. 17 details a procedure 1700 for performing the flip. (Emphasis added).
>
> [0130] Next the source (which after the negotiation becomes the new destination), creates an empty inode map file with one entry for each inode in the source qtree (step 1715). The new destination then initializes a counter with (in this example) N=1(step 1720). N is the variable representing the inode count on the new destination qtree.
>
> [0131] In step 1725, the new destination looks up the Nth inode from the entries associated with the old destination in the stored inode map file (i.e. the map from the old destination/new source). Next, the new destination determines if such an entry exists (decision step 1730). If no entry exists, then a zero entry is created in the new inode map file, representing that the Nth inode of the new source (old destination) is not allocated. However, if there exists an Nth inode of the new source/old destination, then the decision step 1730 branches to step 1740, and creates a new entry in the new inode map file (created in step 1715). The new entry maps the new source (old destination) Nth inode to the proper new destination (old source) inode. Note, in an alternate embodiment, the new inode map is provided with a full field of zero entries before the mapping begins, and the creation of a "zero entry," in this case should be taken broadly to include leaving a preexisting zero entry in place in the inode map.

Thus, Federwisch describes a technique for performing a Inode Map Flip where the new destination (e.g., the original source) first looks up the Nth inode from the entries of the inode map of the new source (e.g., the original destination). If no entry exists in the inode map of the new source, the Federwisch system places a zero entry in the new inode map of the new destination (e.g., the original source). Said differently, the occurrence of a non-existent entry in the inode map at the source (e.g., the original destination) in Federwisch causes the Federwisch system to place a zero in the new inode map entry at the destination (original source). In sharp contrast, the occurrence of a "hole" (e.g., a zero value) in the Applicant's technique causes the Applicant's technique "***to examine a corresponding virtual block number pointer in the backing store***". The Applicant notes that the occurrence of a non-existent entry in Federwisch **does not** cause the Federwisch system to examine a corresponding vbn pointer in a different entity. Instead, Federwisch simply places a zero value in the new inode map indicating that that inode is not in use. The Applicant notes that placing a zero value that indicates that the inode is not in use is very different than the Applicant's claimed "hole" (e.g., a zero value) that indicates that the a corresponding virtual block in the backing store should be examined.

Further, in the Applicant's technique, the "holes" (e.g., zero values) of the writable vdisk are replaced by reference to **data** at the corresponding locations of the backing store. The Applicant notes that a non-existent entry in Federwisch does not cause the Federwisch system to replace that non-existent entry with a reference to data from a different entity (e.g., the Applicant's claimed backing store). Instead, the existence of a non-existent entry in Federwisch causes the Federwisch system to place a zero value in the new inode map (e.g., to synchronize the inode maps at the source and destination). That is, Federwisch makes no mention of encountering a non-existent entry and replacing that non-existent entry with a reference to data. As such, the Applicant submits that Federwisch may not fairly be interpreted to teach or suggest the Applicant's claimed "***replacing each field having a hole in the writable vdisk indirect block with a new pointer to the data block referenced by the corresponding backing store indirect block field to update the writable vdisk to reference both the data which is unchanged***

since the writable vdisk was created and the data which has been changed since the writable vdisk was created."

Further, the Applicant respectfully submits that the deficiencies of Federwisch are not remedied by a combination with Edwards. Instead, Edwards states that if a pointer is directed to an invalid block, the pointer is cleared. *See* Edwards, paragraph [0048].

Accordingly, the Applicant respectfully submits that a combination of Federwisch and Edwards is legally insufficient to render the present claims unpatentable under 35 U.S.C §103(a) because of the absence in Federwisch and Edwards of the Applicant's claimed "***searching each field of the writable vdisk indirect block for a hole***" and " ... ***where each hole instructs the data storage system to examine a corresponding virtual block number pointer in the backing store***" and "***replacing each field having a hole in the writable vdisk indirect block with a new pointer to the data block referenced by the corresponding backing store indirect block field to update the writable vdisk to reference both the data which is unchanged since the writable vdisk was created and the data which has been changed since the writable vdisk was created.***"

At paragraphs 27 – 37 of the Office Action claims 2, 4, 5, 11, 12, 24, 26, 27, 32, 34, and 35 were rejected under 35 U.S.C §103(a) as being unpatentable over Federwisch in view of Edwards, in further view of Haskin et al., U.S. Publication No. 2003/0158863 (hereinafter "Haskin").

The Applicant notes that claims 2, 4, 5, 11, 12, 24, 26, 27, 32, 34, and 35 are dependent claims that depend from independent claims believed to be in condition for allowance. Accordingly, claims 2, 4, 5, 11, 12, 24, 26, 27, 32, 34, and 35 are believed to be in condition for allowance due to their dependency as well as for other separate reasons.

Conclusion

In summary, all the independent claims are believed to be in condition for allowance and therefore all dependent claims that depend there from are believed to be in condition for allowance. The Applicant respectfully solicits favorable action.

Please charge any additional fee occasioned by this paper to our Deposit Account No. 03-1237.

Respectfully submitted,

/Omar M. Wadhwa/
Omar M. Wadhwa
Reg. No. 64,127
CESARI AND MCKENNA, LLP
88 Black Falcon Avenue
Boston, MA 02210-2414
(617) 951-2500